JP9091143

Publication Title:

METHOD AND DEVICE FOR PROCESSING DATA

Abstract:

Abstract of JP 9091143

(A) Translate this text PROBLEM TO BE SOLVED: To simplify the configuration of clients and to reduce costs. SOLUTION: A server 1 and clients 2-1 and 2-2 are connected through a network 3. At the server 1, execution environments 12-1 and 12-2 corresponding to execution environments 22-1 and 22-2 provided at the clients 2-1 and 2-2 are prepared in advance. When prescribed objects 24-1 and 24-2 are required in application programs 21-1 and 21-2 of the clients 2-1 and 2-2, objects 14-1 and 14-2 in application programs 11-1 and 11-2 of the server 1 are down loaded.

---

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－91143

(43)公開日　平成９年(1997)４月４日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ			技術表示箇所
Ｇ０６Ｆ	9/44	５３０		Ｇ０６Ｆ	9/44	５３０Ｊ	
	9/445				13/00	３５１Ｈ	
	13/00	３５１				３５７Ｚ	
		３５７			9/06	４２０Ｊ	

審査請求　未請求　請求項の数15　ＯＬ　（全 20 頁）

(21)出願番号　特願平8－67881

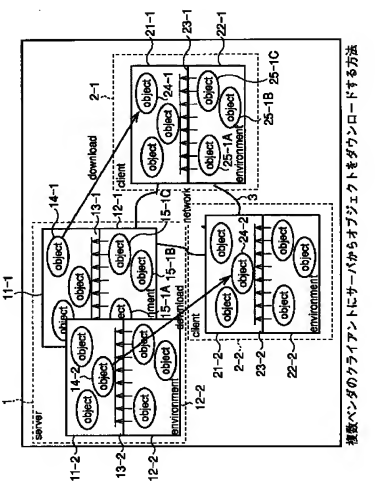
(22)出願日　平成８年(1996)３月25日

(31)優先権主張番号　特願平7－178625
(32)優先日　平７(1995)７月14日
(33)優先権主張国　日本（ＪＰ）

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号
(72)発明者　横手　靖彦
東京都品川区北品川６丁目７番35号　ソニー株式会社内
(74)代理人　弁理士　稲本　義雄

(54)【発明の名称】　データ処理方法および装置

(57)【要約】
【課題】　クライアントの構成を簡略化し、低コスト化を可能にする。
【解決手段】　サーバ１とクライアント２－１，２－２をネットワーク３を介して接続する。サーバ１には、クライアント２－１，２－２が有する実行環境２２－１，２２－２に対応する実行環境１２－１，１２－２を予め用意するようにする。クライアント２－１，２－２のアプリケーションプログラム２１－１，２１－２において、所定のオブジェクト２４－１，２４－２が必要になったとき、サーバ１のアプリケーションプログラム１１－１，１１－２のオブジェクト１４－１，１４－２をダウンロードする。



【特許請求の範囲】
【請求項１】　複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成される、前記アプリケーションプログラムの動作を規定する実行環境と、前記アプリケーションプログラムと前記実行環境の間のインタフェースを規定するアプリケーションプログラムインタフェースとを備えるサーバと、
前記サーバより前記アプリケーションプログラムをダウンロードするクライアントと、
を備えるデータ処理システムにおけるデータ処理方法において、
前記サーバは、前記クライアントに前記アプリケーションプログラムをダウンロードするとき、前記クライアントが、ダウンロードする前記アプリケーションプログラムの実行環境を有するか否かを検査し、
その検査結果に対応して前記アプリケーションプログラムを前記クライアントにダウンロードすることを特徴とするデータ処理方法。
【請求項２】　前記クライアントが、ダウンロードする前記アプリケーションプログラムの実行環境を有しないとき、前記サーバから前記クライアントにオブジェクトをダウンロードし、前記サーバにおける前記実行環境と同一の実行環境を前記クライアントに構築し、その後、前記サーバから前記クライアントに前記アプリケーションプログラムをダウンロードすることを特徴とする請求項１に記載のデータ処理方法。
【請求項３】　前記アプリケーションプログラムのオブジェクトは、並行オブジェクトであることを特徴とする請求項１に記載のデータ処理方法。
【請求項４】　前記サーバは、前記アプリケーションプログラムの実行に必要な前記実行環境のオブジェクトをインクリメンタルにダウンロードすることを特徴とする請求項１に記載のデータ処理方法。
【請求項５】　前記クライアントの実行環境には、最低限の標準的機能として、
前記サーバから前記クライアントへオブジェクトをダウンロードするダウンロード機能と、
前記実行環境の互換性を検査する検査機能と、
前記実行環境を構築する構築機能とを具備させ、
必要に応じてその他の機能をダウンロードさせることを特徴とする請求項２に記載のデータ処理方法。
【請求項６】　前記クライアントには、最低限の標準的機能として、
デバイスドライバのための実行環境と、
システムオブジェクトのための実行環境と、
実行環境のための実行環境とを具備させ、
必要に応じてその他の機能をダウンロードさせることを特徴とする請求項２に記載のデータ処理方法。
【請求項７】　前記デバイスドライバは、インプットドライバ、タイマドライバ、スクリーンドライバを含み、前記システムオブジェクトは、インプットハンドラ、ブートプロトコル、メモリマネージャの各オブジェクトを含むことを特徴とする請求項６に記載のデータ処理方法。
【請求項８】　前記サーバと前記クライアントは、前記クライアントの前記実行環境の互換性を検査するために、前記オブジェクトと前記実行環境について記述したデスクリプションを含むフィーチャストラクチャを授受することを特徴とする請求項１に記載のデータ処理方法。
【請求項９】　複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成される、前記アプリケーションプログラムの動作環境を規定する実行環境と、前記アプリケーションプログラムと前記実行環境の間のインタフェースを規定するアプリケーションプログラムインタフェースとを備え、クライアントにアプリケーションプログラムをダウンロードさせるデータ処理装置において、
前記クライアントに前記アプリケーションプログラムをダウンロードするとき、前記クライアントが、ダウンロードする前記アプリケーションプログラムの実行環境を有するか否かを検査する検査手段と、
前記検査手段の検査結果に対応して前記アプリケーションプログラムを前記クライアントにダウンロードするダウンロード手段とを備えることを特徴とするデータ処理装置。
【請求項１０】　複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成される、前記アプリケーションプログラムの動作環境を規定する実行環境と、前記アプリケーションプログラムと前記実行環境の間のインタフェースを規定するアプリケーションプログラムインタフェースとを備え、サーバからアプリケーションプログラムをダウンロードするデータ処理装置において、
前記サーバから前記アプリケーションプログラムをダウンロードするとき、ダウンロードする前記アプリケーションプログラムの実行環境に関する告知を行う告知手段と、
前記告知手段の告知に対応して前記サーバから前記アプリケーションプログラムをダウンロードするダウンロード手段とを備えることを特徴とするデータ処理装置。
【請求項１１】　複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成され、前記アプリケーションプログラムに対して実行環境を提供するシステムオブジェクトからなるデータ処理装置において、
中間コードに変換された前記アプリケーションプログラムを解釈し、実行する第１の実行手段と、
前記中間コードを動的にコンパイルし、バイナリコードを生成するバイナリコード生成手段と、

前記バイナリコードおよび前記システムオブジェクトを実行する第２の実行手段とを備えることを特徴とするデータ処理装置。

【請求項１２】　前記第１の実行手段は、前記第２の実行手段を介して前記システムオブジェクトを実行することを特徴とする請求項１１に記載のデータ処理装置。

【請求項１３】　前記アプリケーションプログラムの実行状態を示す第１の構造体と、

前記システムオブジェクトが提供する実行環境のアプリケーションプログラムインタフェースを決定する第２の構造体とをさらに備え、

前記第１の実行手段および前記第２の実行手段は、前記第１の構造体と前記第２の構造体に基づいて、前記アプリケーションプログラムの実行を制御することを特徴とする請求項１１に記載のデータ処理装置。

【請求項１４】　前記第１の実行手段は、前記オブジェクトの所定のものにより発行され、所定のオペランドで指定された所定のオペレーションを実行する第１の命令と、前記オペレーションを実行した後、前記命令を発行した前記オブジェクトに制御を戻す第２の命令とを有することを特徴とする請求項１１に記載のデータ処理装置。

【請求項１５】　複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成され、前記アプリケーションプログラムに対して実行環境を提供するシステムオブジェクトからなるデータ処理装置におけるデータ処理方法において、

中間コードに変換された前記アプリケーションプログラムを解釈し、実行する方法と、前記中間コードを動的にコンパイルし、生成されたバイナリコードを実行する方法とにより、前記アプリケーションプログラムを実行することを特徴とするデータ処理方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明はデータ処理方法および装置に関し、特に構成を簡略化し、低コスト化することができるようにした、データ処理方法および装置に関する。

【０００２】

【従来の技術】最近、パーソナルコンピュータが普及してきた。このパーソナルコンピュータでネットワークを介して所定のサーバにアクセスし、所定の情報を得ることができる。

【０００３】このようなパーソナルコンピュータにおいて各種の処理を行うには、アプリケーションプログラムが必要となる。そこで、各ユーザは、そのパーソナルコンピュータのＯＳに合ったアプリケーションプログラムを購入し、インストールして用いるようにしている。

【０００４】

【発明が解決しようとする課題】このように、ＯＳが異なると、アプリケーションプログラムも異なるものとなるため、各ユーザは、自分のＯＳに合ったアプリケーションプログラムを選択して購入する必要がある。また、アプリケーションプログラムを提供する側（アプリケーションプログラムを設計する側）においても、実質的に同一の処理を行うアプリケーションプログラムを複数（ＯＳの数だけ）設計し、用意しなければならず、多くの労力が必要になると同時に、コスト高となる課題があった。

【０００５】また、同様のことが、同一のＯＳにおける各アプリケーションプログラムにおいても発生していた。すなわち、１つのアプリケーションプログラムと、それとは異なる他のアプリケーションプログラムとが、同一のＯＳ上において動作するものであったとしても、２つのアプリケーションプログラムはそれぞれ別個に設計しなければならず、結果的に、１つのアプリケーションプログラムを提供するのに必要な労力とコストが高くなる課題があった。

【０００６】本発明はこのような状況に鑑みてなされたものであり、１つのアプリケーションプログラムを、簡単かつ低コストで提供できるようにするものである。

【０００７】

【課題を解決するための手段】請求項１に記載のデータ処理方法は、サーバが、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査し、その検査結果に対応してアプリケーションプログラムをクライアントにダウンロードすることを特徴とする。

【０００８】請求項９に記載のデータ処理装置は、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査する検査手段と、検査手段の検査結果に対応してアプリケーションプログラムをクライアントにダウンロードするダウンロード手段とを備えることを特徴とする。

【０００９】請求項１０に記載のデータ処理装置は、サーバからアプリケーションプログラムをダウンロードするとき、ダウンロードするアプリケーションプログラムの実行環境に関する告知を行う告知手段と、告知手段の告知に対応してサーバからアプリケーションプログラムをダウンロードするダウンロード手段とを備えることを特徴とする。

【００１０】請求項１１に記載のデータ処理装置は、中間コードに変換されたアプリケーションプログラムを解釈し、実行する第１の実行手段と、中間コードを動的にコンパイルし、バイナリコードを生成するバイナリコード生成手段と、バイナリコードおよびシステムオブジェクトを実行する第２の実行手段とを備えることを特徴とする。

【００１１】請求項１５に記載のデータ処理方法は、中間コードに変換されたアプリケーションプログラムを解釈し、実行する第１の方法と、中間コードを動的にコンパイルし、生成されたバイナリコードを実行する第２の方法とにより、アプリケーションプログラムを実行することを特徴とする。

【００１２】請求項１に記載のデータ処理方法においては、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かをクライアントとサーバ間で検査し、その検査結果に対応してアプリケーションプログラムをクライアントにダウンロードする。

【００１３】請求項９に記載のデータ処理装置においては、検査手段が、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査し、ダウンロード手段が、検査手段の検査結果に対応してアプリケーションプログラムをクライアントにダウンロードする。

【００１４】請求項１０に記載のデータ処理装置においては、告知手段が、サーバからアプリケーションプログラムをダウンロードするとき、ダウンロードするアプリケーションプログラムの実行環境に関する告知を行い、ダウンロード手段が、告知手段の告知に対応してサーバからアプリケーションプログラムをダウンロードする。

【００１５】請求項１１に記載のデータ処理装置においては、第１の実行手段が、中間コードに変換されたプログラムを解釈し、実行し、バイナリコード生成手段が、中間コードを動的にコンパイルし、バイナリコードを生成し、第２の実行手段が、バイナリコードを実行する。例えば、動的コンパイルが困難な場合、中間コードを逐次解釈し、実行することができる。

【００１６】請求項１５に記載のデータ処理方法においては、中間コードに変換されたプログラムを解釈し、実行する第１の方法と、中間コードを動的にコンパイルし、生成されたバイナリコードを実行する第２の方法とにより、アプリケーションプログラムを実行する。例えば、動的コンパイルが困難な場合、中間コードを逐次解釈し、実行することができる。

【００１７】

【発明の実施の形態】本発明のデータ処理方法を適用するシステム構成例を図１に示す。システムはサーバ１（データ処理装置）、クライアント２（データ処理装置）、ネットワーク３から構成されている。

【００１８】すなわち、この実施例においては、サーバ１は２つのアプリケーションプログラムを有し、一方のアプリケーションプログラム１１－１は、それを実行する環境を規定する実行環境１２－１と、アプリケーションプログラム１１－１と実行環境１２－１との間のインタフェースを構成するアプリケーションプログラムインタフェース（ＡＰＩ）１３－１を有している。

【００１９】アプリケーションプログラム１１－１は、複数のオブジェクト１４－１により構成され、また、実行環境１２－１も、複数のオブジェクト１５－１により構成されている。

【００２０】同様に、アプリケーションプログラム１１－２は、その環境を規定する実行環境１２－２と、アプリケーションプログラム１１－２と実行環境１２－２との間のインタフェースとして機能するＡＰＩ１３－２を有している。

【００２１】このアプリケーションプログラム１１－２も、複数のオブジェクト１４－２により構成され、また、実行環境１２－２も複数のオブジェクト１５－２により構成されている。

【００２２】同様に、クライアント２も２つのアプリケーションプログラムを有し、一方のアプリケーションプログラム２１－１は、その環境を規定する実行環境２２－１と、アプリケーションプログラム２１－１と実行環境２２－１との間のインタフェースであるＡＰＩ２３－１を有している。アプリケーションプログラム２１－１と実行環境２２－１は、それぞれ複数のオブジェクト２４－１と２５－１により構成されている。

【００２３】同様に、アプリケーションプログラム２１－２も、その環境を規定する実行環境２２－２とＡＰＩ２３－２とを有し、アプリケーションプログラム２１－２と実行環境２２－２は、それぞれ複数のオブジェクト２４－２と２５－２により構成されている。

【００２４】ここで、各オブジェクトはすべて、他のオブジェクトと並行して処理を実行する並行オブジェクトとして定義されている。１つのＡＰＩの集合は１つの実行環境によって与えられるので、サーバ１、クライアント２中には、複数のＡＰＩが存在することになる。

【００２５】図１に示すように、また図２に拡大して示すように、アプリケーションプログラム１１を複数のオブジェクト１４の集まりで構成する。また、オブジェクト１４を並行オブジェクトとして構成することにより、アプリケーションプログラム１１は並行処理されることになり、実行速度の向上に貢献する。また、オブジェクト１４は置き換えの単位でもあるので、動作にバグのあるオブジェクト、性能上問題のあるオブジェクトなどを、誤りのないオブジェクトで置き換えることによって、アプリケーションプログラム１１全体を作り替えることなく、問題点を解決できる。さらに、オブジェクト１４を部品とし、既存のアプリケーションプログラムの部品としてのオブジェクトを組み合わせることによって、簡単に新しいアプリケーションプログラムを作ることができる。

【００２６】ここで、アプリケーションプログラムとは、１つのサービス単位である。例えば、サーバ１から

の映像データを単に表示しているアプリケーションプログラム、ＶＣＲ機能を用いて映像データを検索しているアプリケーションプログラム、メニューによってサービスを選択しているアプリケーションプログラム、ホームショッピングのアプリケーションプログラム、ホームショッピングと連結した家計簿アプリケーションプログラム、税金計算アプリケーションプログラム等である。

【００２７】アプリケーションプログラム間でオブジェクトを共有することによって、操作性に共通点を持たせることができる。例えば、家計簿でデータを入力しているエディタと、ホームショッピングでのデータ入力エディタを共通化することができる。

【００２８】次に並行オブジェクト（ｃｏｎｃｕｒｒｅｎｔ　ｏｂｊｅｃｔ）について説明する。並行オブジェクトの構成を図３に示す。並行オブジェクトであるオブジェクト１４は、外部に公開されたメソッドエントリのテーブル１４Ａ、メソッドの本体１４Ｂ、オブジェクトの状態を保持するメモリ領域１４Ｃ、メソッドを実行する単一のスレッド１４Ｄを有している。並行オブジェクトには１つの実行コンテキスト（スレッドと呼んでも良い）のみが存在する。従って並行オブジェクトは、１つのメッセージを受信して、その処理中には、新たに到着したメッセージの処理は、現在の実行が終了するまで行わない。

【００２９】このように、オブジェクト内にスレッドを１個だけ配置するようにすると、次の利点が得られる。

【００３０】（１）複数のアクティビティ間の同期を気にする必要がない。すなわち、共有データが存在する場合に、セマフォア等の同期のための命令を用いて、共有データに対するアクセスを順序付けるといったことを行う必要がなくなる。換言すれば、オブジェクトへのメッセージ送信が、その順序付けを含んでいることになる。

【００３１】（２）そのため、同期の取り方のミスによるプログラム誤りが生じなくなると同時に、そのオブジェクトの再利用可能性が高まる。

【００３２】（３）例えばデバイスドライバを本方式で作成することによって、多くの場合に生じる、同期誤りを防ぐことができる。

【００３３】（４）また、デバイスドライバの置き換えによる同期誤りを防ぐことができるので、安全にデバイスドライバを置き換えることができる。

【００３４】（５）デバイスドライバの、実際にハードウェアを制御する部分以外の部分を、ＯＳと独立に作成することができる。これによって、従来から開発のかなりの時間を占めていたデバイスドライバを、共通に開発することが可能になるので、開発期間の短縮につながる。

【００３５】（６）オブジェクト間の実行制御に関する記述をアプリケーションプログラムの記述から除くことができる。例えば、マルチスレッドを用いた手法では（複数のスレッドを用いる場合には）、スレッドの実行制御をアプリケーションプログラム中にプログラムする必要があるために、スレッドのプログラミング環境が変更になると、アプリケーションプログラムを書き換える必要がある。しかしながら、スレッドが１個の場合には、アプリケーションプログラムにこの部分を記述する必要がないので、実行制御方法が変わってもアプリケーションプログラムを書き直す必要がない。並行オブジェクトのその実行環境への最適な実行制御方法は、システムが、後述するオブジェクトの動的拡張の原理を用いて提供する。

【００３６】（７）従って、アプリケーションプログラムを記述する場合には、並列処理を考える必要がない。並行オブジェクトが並列処理の単位であるので、並行オブジェクトをプログラムしていけば、後はシステムが自動的にそのハードウェアに最適な実行制御を行って並列処理が行われる。従来の手法では、いくつプロセスを生成するか、いくつスレッドを生成するか、といったことをプログラミング時に指定しなければならず、この指定は、ハードウェアの性能を考慮しないと、そのアプリケーションプログラムは特定のハードウェア専用のものになってしまうが、本方式によれば、そのようなことはない。

【００３７】本システムにおいては、オブジェクトは必要に応じてダウンロードされる。複数ベンダのクライアント２に対してサーバ１からオブジェクトをダウンロードする場合のシステム例を図４に示す。サーバ１上には、それぞれのベンダ用のクライアントＡＰＩ１３（１３－１，１３－２）が、実行環境１２（１２－１，１２－２）によって実現されている。

【００３８】オブジェクトをクライアント２（２－１，２－２）にダウンロードするとき、クライアント２上に、サーバ１上の実行環境１２と同じ実行環境２２（２２－１，２２－２）が存在するか否かを調べ、存在する場合には、オブジェクトをダウンロードする。存在しない場合には、クライアント２上にサーバ１上の実行環境と同一の実行環境を構築した後、ダウンロードする。

【００３９】例えば、図４において、サーバ１のアプリケーションプログラム１１－１のオブジェクト１４－１を、クライアント２－１のアプリケーションプログラム２１－１のオブジェクト２４－１としてダウンロードする場合、クライアント２－１の実行環境２２－１に、サーバ１の実行環境１２－１のオブジェクト１５－１Ａに対応するオブジェクト２５－１Ａが必要であるとき、例えば、実行環境１２－１のオブジェクト１５－１Ｂ（検査手段）は、実行環境２２－１のオブジェクト２５－１Ｂ（告知手段）にフィーチャストラクチャ（後述する）を問い合わせる。そして、その回答に対応して、実行環境１２－１のオブジェクト１５－１Ｃ（ダウンロード手段）と実行環境２２－１のオブジェクト２５－１Ｃ（ダ

ウンロード手段）は、実行環境１２－１のオブジェクト１５－１Ａと１５－１Ｂを、実行環境２２－１のオブジェクト２５－１Ａと２５－１Ｂとしてダウンロードさせる。

【００４０】従来の手法では、ダウンロードするオブジェクトは、クライアントのＡＰＩに合わせて作成する必要があった。例えば、クライアントがＵＮＩＸシステムの場合には、サーバ上では同じＵＮＩＸシステムを用いるか、あるいは何らかのクロス開発環境を構築してオブジェクトを作成する必要がある。もし、サーバとクライアントが同じ実行環境を備えなければいけない場合には、クライアント装置は一般に高価な計算資源を備える必要がある。例えば、専用の実行環境を備える場合と比べて、より多くのメモリを備える必要がある。また、十分な実行速度を保証するために、高速のＣＰＵ（Central Processing Unit）を備える必要がある。これは、装置のコスト増につながる。

【００４１】これに対して、本システムでは、アプリケーションのダウンロードと同時に、その実行環境をもダウンロードすることで、この問題を解決する。すなわち、クライアント２で現在必要とする実行環境２２のみをクライアント２に構築することによって、不必要な資源をクライアント２に用意しなくてもすむようになる。例えば、クライアント２が３Ｄのグラフィックスを必要としない場合には、そのライブラリは必要なくなる。

【００４２】また、クライアント２がＶＯＤ（Ｖｉｄｅｏ　Ｏｎ　Ｄｅｍａｎｄ）で映画を見ている場合には、ユーザとのインタラクションのためのサービス（映画を見るときは不要となるサービス）をクライアントから一時削除することによって、その分の計算資源を他の仕事に割り振ることができる。例えば、その資源を、サーバ１からの映像データのプリフェッチ用のバッファに使うことができる。インタラクションのためのサービスは、それが必要になった時点でサーバ１からダウンロードされる。

【００４３】本システムでダウンロードされるオブジェクトとしては次のものが考えられる。

【００４４】（１）すべてのアプリケーションプログラム

【００４５】（２）クライアントが備えるハードウェア資源を制御するためのデバイスドライバ群（例えば、ＭＰＥＧドライバ、ＡＴＭドライバ、画像制御ドライバ等）

【００４６】（３）アプリケーションプログラムに対してシステムサービスを提供するオブジェクト群（例えば、ＶＣＲコマンド管理、ストリーム管理、実時間スケジューラ、メモリ管理、ウィンドウ管理、ダウンロード制御、通信プロトコル管理、実行管理等）

【００４７】これらを組み合わせて、アプリケーションプログラムに対して最適な実行環境をクライアント上に構築する。

【００４８】サーバ１は、映像データやアプリケーションプログラムを送出する装置であったり、クライアント２にネットワーク３を通して情報を送出する装置である。一方クライアント２は、サーバ１からの情報を処理する装置であり、常にネットワーク３と接続されている必要はない。実行環境はアプリケーションプログラム毎に与えることができるので、アプリケーションプログラム毎に最適な実行環境を用意することができる。

【００４９】従来の手法では、システム構築時にアプリケーションの特性をあらかじめ見積もっておく必要があった。例えば、アプリケーションプログラムが映像データを扱う必要があるときには、そのためのシステムサービス、例えば、実時間スケジューリングや映像データを扱うためのＶＣＲのようなユーザインタフェースを備えている必要がある。また、アプリケーションが３Ｄグラフィックスを用いているならば、そのためのライブラリを備えている必要がある。そのため、システムは肥大になる傾向があった。ＵＮＩＸやＷｉｎｄｏｗｓ（商標）がこの典型例であり、バージョンがあがるごとにシステムが必要とするメモリ量は多くなっていった。本システムでは、アプリケーションの実行のために必要最小限の機能のみを備えていればよく、従来システムのこの問題点を解決する。

【００５０】アプリケーションプログラム１１を複数のオブジェクトの集まりとして構成することにより、また、そのオブジェクトを並行オブジェクトとして実装することにより、オブジェクト単位で並行実行が可能になり、アプリケーションプログラムの実行と同時にオブジェクトをダウンロードすることができる。この時、図５に示すように、アプリケーションプログラムの実行に必要なオブジェクトをインクリメンタルにダウンロードすることにより、ユーザからは、単一のアプリケーションプログラムのロードにかかる時間を隠すことができる。

【００５１】例えば、図５に示すように、サーバ１のアプリケーションプログラム１１のオブジェクト１４－１－１乃至１４－１－１１を、クライアント２のアプリケーションプログラム２１のオブジェクト２４－１－１乃至２４－１－１１としてダウンロードする必要がある場合、ランダムに各オブジェクトをダウンロードするのではなく、アプリケーションプログラム２１を実行する上において、最初に必要なオブジェクト１４－１－１乃至１４－１－３を、オブジェクト２４－１－１乃至２４－１－３として先にダウンロードする。

【００５２】アプリケーションプログラム２１は、さしあたって、この３つのオブジェクトが存在すれば、起動可能であるため、その処理を開始する。そして、その処理が実行されている間に、残りのオブジェクト１４－１－４乃至１４－１－１１をアプリケーションプログラム２１のオブジェクト２４－１－４乃至２４－１－１１と

して、第２乃至第４のダウンロードで順次ダウンロードする。この第２乃至第４のダウンロードにおいても、処理上、先に必要となるオブジェクトから順番にダウンロードする。

【００５３】ユーザは、アプリケーションプログラム２１がオブジェクト２４－１－１乃至２４－１－３がダウンロードされ、その処理が開始された時点において、既にアプリケーションプログラム２１の処理が開始されているため、あたかもすべてのオブジェクトのダウンロードが完了したものと認識することができる。すなわち、ユーザは、１１個のオブジェクトをダウンロードするのに必要な時間より短い、３個のオブジェクトをダウンロードするのに必要な時間だけを意識することになる。換言すれば、ユーザに対して、８個のオブジェクトをダウンロードするための時間を実質的に隠す（意識させないようにする）ことができる。

【００５４】このことは、図４を参照して説明した（また、図１０を参照して後述する）実行環境２２の構築をクライアント２上に行う場合にも当てはまる。この場合は、実行環境２２を構成するオブジェクトのうち、アプリケーションプログラムの実行に必要な部分のみのオブジェクトを先にダウンロードすることによって、実行環境のすべてのオブジェクトをダウンロードする時間をユーザから隠すことができる。この手法は、システムのブートにも応用できる。

【００５５】ここで、インクリメンタルダウンロードとは、アプリケーションプログラムや実行環境を一度にダウンロードしないで、それを構成するオブジェクト単位、あるいは、その一部を必要に応じてダウンロードすることを意味する。従来のパソコン通信におけるアプリケーションプログラムのダウンロードの場合には、圧縮されたアプリケーションプログラムを一気にダウンロードするので、ダウンロードがすべて終わらないと、そのアプリケーションプログラムは利用することができない。また、例えば、今までのシステムのブートでは、システム全体を全部メモリに読み込んでから立ち上がる。ＵＮＩＸのディスクレスワークステーションの場合には、サーバからすべてのＯＳをメモリに読み込んでからシステムが起動するので、読み込みがすべて終了するまで、システムを利用できない。しかしながらインクリメンタルダウンロードすれば、そのようなことがなくなる。

【００５６】この手法は、サーバ１とクライアント２としてのＳＴＢ（Ｓｅｔ　Ｔｏｐ　Ｂｏｘ）に応用して次のような効果がある。まず、ＳＴＢの電源を入れるとすぐに利用できるようになる。現在のパーソナルコンピュータのように、システムが立ち上がるまでいらいらして待つ必要が無くなる。ＳＴＢは家庭用電気製品としての性格が強いので、システムが立ち上がるまでユーザを待たせるのは好ましくない。

【００５７】ＳＴＢの電源が入れられると、ＳＴＢは最初に必要なオブジェクトをダウンロードして実行を始めようとする。ユーザの待ち時間は最初のこのオブジェクトのダウンロード時間のみである。典型的なオブジェクトのダウンロード時間は数ミリ秒から数十ミリ秒であるので、適当なユーザインタフェースを備えることにより、この時間は十分ユーザにとって無視できる時間となる。以降は、システムの立ち上げプロセスの進行に従って、必要なオブジェクトがシステムの立ち上げプロセスと並行にダウンロードされる。

【００５８】また、ＳＴＢは、そのコストの観点から、サーバのような豊富な計算資源が用意されていないので、複数のアプリケーションを同時に実行する場合にも制約が生じる。例えば、ナビゲーションアプリケーションによって、ＶＯＤサービスを選択して、１つの映画を鑑賞することを考えた場合、映画の鑑賞が始まったら、ナビゲーションアプリケーションが占有していた資源（メモリ）を解放して、映画鑑賞アプリケーションのために使うことができる。そして、再び、ナビゲーションアプリケーションが必要になった時点で、その資源（メモリを管理するオブジェクト）をダウンロードする。

【００５９】ここで、「必要になった時点」とは、オブジェクトに対して何らかのメッセージが送られた時点とする。すなわち、一番最初にダウンロードしたオブジェクトが、別のオブジェクトにメッセージを送った時点で、その受信オブジェクトをダウンロードする。オブジェクトの依存、参照関係を利用することによって、次のメッセージを送るオブジェクトをあらかじめダウンロードしておくことができる。これを、アプリケーションの実行と並行に行うことにより、メッセージ通信時におけるダウンロードによる遅延を少なくすることができる。これにより、インクリメンタルダウンローディングの有効性を高めることができる。

【００６０】また、実行環境１２，２２もオブジェクト１５，２５の集合体であり、アプリケーションプログラム１１，２１のオブジェクト１４，２４と同等に操作可能であるので、アプリケーションプログラム１１，２１に特化したダウンロードの順序を制御するメタオブジェクトをオブジェクト１５，２５の１つとして用意する（例えば図４のオブジェクト２５－１Ｃをメタオブジェクトとする）ことができる。これによって、特定のアプリケーションに適した、そのオブジェクトが利用するオブジェクトのダウンロード順を指定することができ、インクリメンタルダウンローディングによるユーザの待ち時間を最小にすることができる。

【００６１】サーバ１からクライアント２へのオブジェクトのダウンロードの機能、それに伴うオブジェクトの実行環境の互換性の検査機能、および実行環境の構築機能は、本システムを実現する上で重要なものであり、本システムを構成するすべての装置（クライアント２）が

最低限有すべき機能とする。本明細書ではこの機能をメタ標準と呼ぶ。このメタ標準により、ＯＳ等の実行環境のＡＰＩは自由に拡張することができるようになり、最小限の標準化とその拡張によって、今後のあらゆるタイプのアプリケーションに対応することができるようになる。

【００６２】例えば、図６に示すシステムでは、各サーバ１－１，１－２、各クライアント２－１，２－２では、独自のＡＰＩを持ったＯＳが稼働している。すなわちクライアント２－１においては、実行環境２２－１に対応して、アプリケーションプログラム２１－１のためのＡＰＩ２３－１（ＡＰＩ＃１）が構成されている。また、クライアント２－２においては、実行環境２２－２によりアプリケーションプログラム２１－２のためのＡＰＩ２３－２（ＡＰＩ＃３）が形成されている。

【００６３】このため、これらのクライアント２－１，２－２に対してプログラムをダウンロードするサーバには、これらのＡＰＩに対応するＡＰＩが予め用意されている。この実施例においては、サーバ１－１において、実行環境１２－１によりアプリケーションプログラム１１－１のためのＡＰＩ１３－１が形成されており、このＡＰＩ１３－１は、クライアント２－１におけるＡＰＩ２３－１（ＡＰＩ＃１）に対応するＡＰＩ（ＡＰＩ＃１）とされている。

【００６４】同様にサーバ１－２において、実行環境１２－３によりアプリケーションプログラム１１－３に対応するＡＰＩ１３－３（ＡＰＩ＃３）が形成されており、このＡＰＩ１３－３がクライアント２－２のＡＰＩ２３－２（ＡＰＩ＃３）に対応している。

【００６５】従って、サーバ１－１，１－２と、クライアント２－１，２－２に、このメタ標準に対応するオブジェクトとして、オブジェクト１５－１Ａ乃至１５－１Ｃ，１５－３Ａ乃至１５－３Ｃ，２５－１Ａ乃至２５－１Ｃおよび２５－２Ａ乃至２５－２Ｃを設けている。その結果、クライアント２－１または２－２に対しては、サーバ１－１または１－２から、メタ標準プロトコルに従って必要なオブジェクトを適宜ダウンロードさせることができる。

【００６６】各クライアントにおけるＯＳを１つのＯＳに標準化しようとするのがこれまでのこの分野における傾向であった。しかしながら、このようなメタ標準を規定し、サーバ側においてのみ、各クライアントのＡＰＩに対応するＡＰＩを具備させることで、標準を決定する必要がなくなる。

【００６７】ＯＳの標準を１つに規定していないことによって、アプリケーションを始め、システムサービスを実現しているオブジェクトは、ＯＳとは独立に構成することが可能になる。すなわち、ある実行環境用に書かれたソフトウェアを、マイグレーションによって別の実行環境用に自動的に再構成すればよいことになる。従来のシステムには、この機能は存在しない。例えば、ＵＮＩＸ用に書かれたソフトウェアはＷｉｎｄｏｗｓ上では、書き直さないと動作しない。アプリケーションレベルのソフトウェアで、この機能を実現するには、ＯＳ依存性を吸収するソフトウェアを導入する必要がある。しかし、デバイスドライバを始めとする、システムサービスを実現するオブジェクトのＯＳの独立性は、本手法を用いて可能になる。

【００６８】このように、オブジェクトをダウンロードするようにすると、クライアント２のオブジェクトは、図７に示すように、動的に変更することができる。すなわち、既存のオブジェクトをクライアントから削除して、新しいオブジェクトをサーバからダウンロードする。

【００６９】例えば、図７の実施例においては、クライアント２のアプリケーションプログラム２１におけるオブジェクト２４Ａが不要となったので、これを削除している。そして新たに必要になったオブジェクト１４Ａをサーバ１からクライアント２に対してダウンロードする。

【００７０】これにより、次のことが可能になる。

【００７１】（１）ソフトウェアをアップデートすることができる。例えば、ハードウェア制御ソフトにバグが見つかった場合に、そのオブジェクトを削除して、新しいオブジェクトに置き換える。家庭電化製品は、コンピュータの専門家ではない、一般消費者が利用するものなので、一部のコンピュータに見られるようなインストーラによるソフトウェアのアップデートは適切ではない。

【００７２】（２）製品のサイクルを長くすることができる。例えば、テレビジョン受像機は毎年のようにモデルチェンジが繰り返されるが、一般消費者は、テレビジョン受像機を毎年買い換るようなことはしない。しかし、本システムにより、ソフトウェアの問題はすべて最新の機能をユーザに提供できることになるので、ソフトウェアの機能拡張によるモデルチェンジをなくすことができる。これは、ＳＴＢの場合にも当てはまる。

【００７３】（３）ユーザインタフェースの好みの変化に対応できる。例えば、初めてその装置を使い始めたユーザには、親切なメニュー形式を提供するが、ユーザがその装置の使い方を習熟するにつれて、直接所望の操作ができるようなダイレクト操作のユーザインタフェースに変更することができる。この場合も、両方の手順をクライアント側に持たせるのではなく、そのときのユーザの熟練度に応じたユーザインタフェースをクライアント側に持たせることができる。これにより、限りあるクライアントの資源を有効に利用することができる。

【００７４】また、オブジェクトをダウンロードすれば、図８に示すようにクライアント２のオブジェクトを動的に拡張することができる。図８の実施例においては、クライアント２におけるアプリケーションプログラ

ム２１－１のオブジェクト２４－１Ｂに対して新たなサービスを受けられるようにするために、実行環境２２－２が生成されている。そして実行環境２２－１から必要なオブジェクト２５－１Ａ，２５－１Ｂが実行環境２２－２にオブジェクト２５－２Ａ，２５－２Ｂとなるように、マイグレート（移転）する。さらに必要なその他のオブジェクト２５－１Ｃ，２５－１Ｄも実行環境２２－２にマイグレートされる。

【００７５】そしてアプリケーションプログラム２１－２には、アプリケーションプログラム２１－１のオブジェクト２４－１Ｂが、オブジェクト２４－２Ｂとなるようにマイグレートされる。

【００７６】このようにして、例えば、実時間スケジューリングの拡張が必要になった場合には、その為の新しい実行環境をクライアントに生成し、必要なオブジェクトをその新しい環境に移動する。オブジェクトには何も変更を加える必要がなく、オブジェクトは実時間スケジューリングのサービスを受けられるようになる。

【００７７】これにより次の効果が得られる。

【００７８】（１）アプリケーションプログラムのオブジェクトに変更を加えないで、新しい機能に対処することが可能になるので、アプリケーションプログラムの寿命が長くなり、再利用可能性が増す。従来の手法では、アプリケーションプログラムにその実行環境に対する依存コードが含まれていたために、実行環境が変わることは、アプリケーションプログラムの書き換えを意味していた。

【００７９】（２）組み込み機器のアプリケーションプログラムの場合には、ユーザインタフェースをはじめとする機器の高機能制御ソフトの部分は、モデルが大きく変わらない限り再利用したい部分であり、また、既存のコードを利用して機能拡張するのみで開発期間を短縮したい部分であるが、この部分に、実行環境への依存コードが含まれていると、再利用のための作業は複雑になる。今までの手法は、この部分に対して何の戦略もなかったが、本方式によって、この作業は自動化、あるいは、最小化できる。

【００８０】本システムは、具体的には次のように応用できる。

【００８１】（１）アプリケーションの信頼性が低い場合に、メモリ保護機能をダウンロードすることによってシステム全体の停止を防ぐことができる。

【００８２】（２）ＳＴＢのベンダ毎に、サービスプロバイダは提供サービスを変えることができる。例えば、映画会社Ａの映画を提供するサービスプロバイダは、その映画を受信するＢ社のＳＴＢとＣ社のＳＴＢとで、画像の質を変えることができる。

【００８３】（３）ＳＴＢに送られてくるＡ／Ｖデータの圧縮方式やユーザの画質の好みに応じて、システムの処理方式を変更する。例えば、ＭＰＥＧデータの場合とＪＰＥＧデータでは画質を調整するときの仕方が異なるので、システムの処理方式を変更する必要があるが、本システムでは、データ形式に従って動的に処理方式を選択できる。

【００８４】このように、システムのほとんどの機能はサーバ１からダウンロードすることができるので、あらかじめクライアント２に様々の機能を持たせる必要が無く、最小限の機能を持たせればよい。図９に本システムにおけるクライアント２が有する最小限の機能を示す。すなわち、クライアント２には、デバイスドライバのための実行環境２２－１、システムオブジェクトのための実行環境２２－２、実行環境の為の実行環境２２－３を、最低限形成する。

【００８５】あらかじめ存在させる必要のあるデバイスドライバは、入力を処理するｉｎｐｕｔ　ｄｒｉｖｅｒ、時間を管理するｔｉｍｅｒ　ｄｒｉｖｅｒ、表示を制御するｓｃｒｅｅｎ　ｄｒｉｖｅｒの各オブジェクト２４－１Ａ，２４－１Ｂ，２４－１Ｃであり、システムオブジェクトは、入力を管理するｉｎｐｕｔ　ｈａｎｄｌｅｒ、起動を管理するｂｏｏｔ　ｐｒｏｔｏｃｏｌ、記憶を管理するｍｅｍｏｒｙ　ｍａｎａｇｅｒの各オブジェクト２４－２Ａ，２４－２Ｂ，２４－２Ｃである。より高機能のデバイスドライバ、システムオブジェクトはサーバ１からダウンロードされる。

【００８６】図１０は、サーバの送り出すアプリケーションプログラム（ビデオ、ゲーム、ショッピング等）に適したクライアント環境を動的に構成する実施例を表している。図１０では、ショッピングのアプリケーションプログラム１１－２をサーバ１からダウンロードするために、ショッピング用の実行環境２２－４がクライアント２－２に構成されている。また、ショッピングアプリケーションプログラムから映画アプリケーションプログラムに、クライアント２－２がアプリケーションプログラムを切り替えた場合には、映画アプリケーションプログラム２１－３の実行環境２２－３がクライアント２－２に構成され、サーバ１から映画アプリケーションプログラム１１－１がダウンロードされる。

【００８７】例えば、次のような処理が考えられる。

【００８８】（１）ユーザが映画を選択しているときこの時には、所望の映画を選択することができるように、ナビゲーションアプリケーション１１－３が、例えばクライアント２－１の実行環境２２－１にダウンロードされ、それに対する必要な環境として、ウィンドウ管理、ユーザからの入力管理等のオブジェクト１５－３がオブジェクト２５－１としてダウンロードされる。

【００８９】（２）ユーザが映画を鑑賞しているときこの時には、サーバ１－１の実行処理１２－１から、ビデオストリーム管理、データ先読みバッファ管理、ＶＣＲ機能等のオブジェクト１５－１が、例えばクライアント２－２の実行環境２２－３にオブジェクト２５－３と

してダウンロードされる。

【００９０】以上のように、クライアントの実行環境をダウンロードにより実現するために、本システムでは図１１に示すフィーチャストラクチャ（ｆｅａｔｕｒｅ　ｓｔｒｕｃｔｕｒｅ）を導入する。オブジェクトがサーバ１からクライアント２にダウンロードされる時、このｆｅａｔｕｒｅ　ｓｔｒｕｃｔｕｒｅが検査されて、必要な実行環境がダウンロード先に構成される。

【００９１】すなわち、サーバ１は、クライアント２との間において、第１のフェーズ（ネゴシエーションフェーズ）で、サーバ１側のメタオブジェクト空間とクライアント２側のメタオブジェクト空間との間におけるオブジェクトマイグレーションの可能性に関するネゴシエーションを行う。そして第２のフェーズ（移動フェーズ）において、実際にオブジェクトの転送を行う。

【００９２】オブジェクトマイグレーションはメタレベルのプロセスであり、オブジェクトの内部情報と、必要ならばそれに関連したオブジェクトによって使用中の計算資源を転送するものである。オブジェクトの内部情報は、ディスクリプタ（ｄｅｓｃｒｉｐｔｏｒ）と呼ばれるメタレベルオブジェクトによって表現されている。ディスクリプタは、実際には、オブジェクトを管理しているメタオブジェクトの名前を保持している。一般的なディスクリプタは、オブジェクトのメモリセグメントを管理しているメタオブジェクトの名前、２つ以上のオブジェクトの実行制御をしているメタオブジェクト（スケジューラ）の名前、オブジェクトの名前付けを管理しているメタオブジェクトの名前、等を保持している。

【００９３】第１のフェーズ（ネゴシエーションフェーズ）においては、オブジェクトの移動の可能性が検査される。すなわち、メタオブジェクト空間によっては、オブジェクトマイグレーションが望ましくない場合がある。例えば、デバイスドライバを管理しているメタオブジェクト空間が、オブジェクト（デバイスドライバ）を移動する場合には、クライアント２側にハードウエアデバイスが実際に存在していなければ、その移動を行っても意味が無くなってしまう。また仮想記憶管理機構を利用して、オブジェクトのメモリセグメントを管理しているメタオブジェクトも、移動先に仮想記憶管理機構が存在しなければ、移動を行ったとしてもメモリセグメントを管理することができない。そこで、マイグレーションプロトコルにおいては、次のメソッドが用意されている。

【００９４】Ｆｅａｔｕｒｅ＊　Ｄｅｓｃｒｉｐｔｏｒ：：ＣａｎＳｐｅａｋ（Ｆｅａｔｕｒｅ＊　ｐＦｅａｔｕｒｅ）

【００９５】このメソッドは、クライアント２側のメタオブジェクト空間内のｄｅｓｃｒｉｐｔｏｒに対して、ＣａｎＳｐｅａｋオペレーションを実行する。

【００９６】この時、フィーチャストラクチャが引数として渡され、結果としてクライアント２は、サーバ１に対して自分自身（クライアント２）が受け入れ可能なフィーチャストラクチャを返す。サーバ１側においては、クライアント２側から返されたフィーチャストラクチャを検査することで、そのメタオブジェット空間の互換性を知ることができる。

【００９７】互換性は、完全互換、半互換、および非互換の３つのカテゴリに分類される。

【００９８】完全互換は、オブジェクトが移動後も完全に実行を続けることができる場合を意味する。半互換は、移動後のオブジェクトの実行には、ある制限が加えられることを意味する。そして、非互換は、オブジェクトは移動後には実行を続けることができないことを意味する。

【００９９】非互換の場合には、オブジェクトマイグレーションは行われない。半互換の場合にはマイグレーションを行うか、行わないかをユーザが判断する。実際には、例外がユーザに返されるので、例外処理ルーチンによって、マイグレーションの判断がなされる。完全互換の場合、または半互換の場合であって、オブジェクトマイグレーションが行われるときには、先に返されたフィーチャストラクチャの内容に従って、マイグレーションが行われる。

【０１００】なお、ネゴシエーションフェーズに先だって、次のオペレーションによって空のｄｅｓｃｒｉｐｔｏｒがクライアント２側のメタオブジェクト空間に生成される。

【０１０１】
Ｄｅｓｃｒｉｐｔｏｒ：：Ｄｅｓｃｒｉｐｔｏｒ（）

【０１０２】先のＣａｎＳｐｅａｋメソッドは、このｄｅｓｃｒｉｐｔｏｒに対して送られる。この時、フィーチャストラクチャの情報を元に必要なメタオブジェクトの生成、参照の生成、必要情報の登録が行われる。

【０１０３】第２のフェーズにおけるメタレベルでのプロセスは、転送オブジェクトに対応したメタオブジェクトの移動、あるは転送になる。ここで、メタオブジェクトの移動とは、そのメタオブジェクトがクライアント２側のメタオブジェクト空間に入ること、すなわち、ディスクリプタ（ｄｅｓｃｒｉｐｔｏｒ）から参照されるようになることを意味し、また、メタオブジェクトの転送とは、クライアント２側のメタオブジェクト空間にあるメタオブジェクト（これはｄｅｓｃｒｉｐｔｏｒから参照されている）に対して、メタオブジェクト内のデータをメッセージとして送ることを意味する。

【０１０４】メタオブジェクトの移動および転送に関する実際のオペレーションが、ネゴシエーションフェーズによって得られたフィーチャストラクチャを利用して、この第２のフェーズ（移動フェーズ）において実行される。

【０１０５】移動フェーズにおける実際のメタオブジェ

クトの移動および転送は、次のメソッドによって起動される。
【０１０６】Ｄｅｓｃｒｉｐｔｏｒ＆　Ｄｅｓｃｒｉｐｔｏｒ：：ｏｐｅｒａｔｏｒ＝（Ｄｅｓｃｒｉｐｔｏｒ＆　ｓｏｕｒｃｅ）
【０１０７】Ｄｅｓｃｒｉｐｔｏｒクラスは抽象クラスであり、メタオブジェクトの移動、転送に関する共通のプロトコルを定義している。ｓｏｕｒｃｅで参照されるｄｅｓｃｒｉｐｔｏｒの内容が、このディスクリプタにコピーされる。
【０１０８】実際の手続きは、Ｄｅｓｃｒｉｐｔｏｒクラスのサブクラスとして定義される。
【０１０９】次のメソッドは、メタオブジェクトの転送に関するメソッドである。これらのプロトコルは主にｍｉｇｒａｔｏｒ（メタオブジェクト空間内に含まれる、オブジェクトマイグレーションを担当するメタオブジェクト）によって使用される。
・ＣａｎｏｎｉｃａｌＣｏｎｔｅｘｔ＆　Ｃｏｎｔｅｘｔ：：ａｓＣａｎｏｎｉｃａｌ（）
機械依存のＣｏｎｔｅｘｔ構造体を、機械非依存の形式に変換する。このプロトコルは、ｆｅａｔｕｒｅ　ｓｔｒｕｃｔｕｒｅによってＣｏｎｔｅｘｔの直接変換が不可能である、と示されたときに実行される。
・Ｃｏｎｔｅｘｔ＆　Ｃｏｎｔｅｘｔ：：ｏｐｅｒａｔｏｒ＝（Ｃｏｎｔｅｘｔ＆　ｓｏｕｒｃｅ）
・Ｃｏｎｔｅｘｔ＆　Ｃｏｎｔｅｘｔ：：ｏｐｅｒａｔｏｒ＝（ＣａｎｏｎｉｃａｌＣｏｎｔｅｘｔ＆　ｓｏｕｒｃｅ）
現在（ｔｈｉｓが参照する）Ｃｏｎｔｅｘｔをｓｏｕｒｃｅが参照するＣｏｎｔｅｘｔで初期化する。
・ＣａｎｏｎｉｃａｌＳｅｇｍｅｎｔ＆　Ｓｅｇｍｅｎｔ：：ａｓＣａｎｏｎｉｃａｌ（）
機械依存のＳｅｇｍｅｎｔ構造体を、機械非依存の形式に変換する。このプロトコルは、ｆｅａｔｕｒｅ　ｓｔｒｕｃｔｕｒｅによってＣｏｎｔｅｘｔの直接変換が不可能である、と示されたときに実行される。
・Ｓｅｇｍｅｎｔ＆　Ｓｅｇｍｅｎｔ：：ｏｐｅｒａｔｏｒ＝（Ｓｅｇｍｅｎｔ＆　ｓｏｕｒｃｅ）
・Ｓｅｇｍｅｎｔ＆　Ｓｅｇｍｅｎｔ：：ｏｐｅｒａｔｏｒ＝（ＣａｎｏｎｉｃａｌＳｅｇｍｅｎｔ＆　ｓｏｕｒｃｅ）
現在のｔｈｉｓ（が参照する）Ｓｅｇｍｅｎｔをｓｏｕｒｃｅが参照するＳｅｇｍｅｎｔで初期化し、必要なメモリ領域をコピーする。
【０１１０】図１１は、フィーチャストラクチャの構造を表している。同図に示すように、ｏｂｊｅｃｔ　ｄｅｓｃｒｉｐｔｉｏｎとｅｎｖｉｒｏｎｍｅｍｔ　ｄｅｓｃｒｉｐｔｉｏｎのポインタが、エントリーに記述されている。
【０１１１】ｏｂｊｅｃｔ　ｄｅｓｃｒｉｐｔｉｏｎのポインタで指示されているストラクチャには、ｏｂｊｅｃｔ　ｎａｍｅが記述され、さらに、ｅｎｖｉｒｏｎｍｅｎｔ　ｄｅｓｃｒｉｐｔｉｏｎのポインタで指示されているストラクチャと同一のストラクチャのポインタが記述されている。そしてさらに、このオブジェクトの資源要求（ｒｅｓｏｕｒｃｅ　ｒｅｑｕｉｒｅｍｎｅｎｔ　ｏｆ　ｔｈｉｓ　ｏｐｊｅｃｔ）が記述されている。
【０１１２】また、ｅｎｖｉｒｏｎｍｅｎｔ　ｄｅｓｃｒｉｐｔｉｏｎのポインタで指示されているストラクチャには、ｅｎｖｉｒｏｎｍｅｎｔ　ｎａｍｅが記述され、さらに、そのクライアントのハードウエアの資源情報（ｒｅｓｏｕｒｃｅ　ｉｎｆｏｒｍａｔｉｏｎ　ｏｆ　ｃｌｉｅｎｔ　ｈａｒｄｗａｒｅ）、この実行環境の資源要求（ｒｅｓｏｕｒｃｅ　ｒｅｑｕｉｒｅｍｅｎｔ　ｏｆ　ｔｈｉｓ　ｅｎｖｉｒｏｎｍｅｎｔ）、およびこの実行環境を構成するメタオブジェクトのリスト（ｌｉｓｔ　ｏｆ　ｍｅｔａｏｂｊｅｃｔｓ　ｃｏｎｓｔｉｔｕｔｉｎｇ　ｅｎｖｉｒｏｎｍｅｎｔ）が記述されている。
【０１１３】フィーチャストラクチャの内容の具体例をあげると、次のようになる。
【０１１４】（１）オブジェクトに関する情報
実時間性
必要プロセッサ量
【０１１５】（２）メタオブジェクトに関する情報
ハードウェアメタオブジェクト
＊プロセッサの種類
＊データ表現形式
セグメントメタオブジェクト
＊大きさ
＊拡大、縮小可能性
＊管理方針
＊レイアウト
コンテキストメタオブジェクト
＊レジスタ情報
＊一時変数量
＊プロセッサ状態
メイラメタオブジェクト
＊メッセージキュー長
＊未処理メッセージ数
＊外部メイラの必要性
＊メッセージの転送方式
外部メイラメタオブジェクト
＊メッセージキュー長
＊未処理メッセージ数
＊プロトコル
スケジューラメタオブジェクト
＊オブジェクト状態
＊スケジューリング方針
依存管理メタオブジェクト

＊保有する外部名数

【０１１６】上述したように、各クライアントが異なるＯＳを有するような場合、このフィーチャストラクチャから各クライアントのＯＳが判定され、サーバは、そのＯＳに対応するオブジェクトをダウンロードさせることになる。

【０１１７】図１２は、本発明のデータ処理システムを応用したＡｐｅｒｔｏｓ（商標）システムの構成例を示している。このＡｐｅｒｔｏｓシステムにおいては、Ａｐｅｒｔｏｓ　Ｍｉｃｒｏ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅ（ＭＶＭ）３１ａ（第１の実行手段）は、Ａｐｅｒｔｏｓ　Ｍｉｃｒｏ　Ｋｅｒｎｅｌ（ＭＫ）３１ｂ（第２の実行手段）とともにあり、Ａｐｅｒｔｏｓシステムの核３１を構成している。核３１を構成するＭＶＭ３１ａは後述する中間コード（Ｉ－ｃｏｄｅ）を解釈実行するが、必要に応じて、ＭＫ３１ｂの機能を用いてｐｅｒｓｏｎａｌｉｔｙオブジェクト（システムオブジェクト）を呼び出すようになされている。

【０１１８】図１２に示した核３１以外の部分は、上述した方法により、例えば、サーバ１よりダウンロードすることができる。

【０１１９】このシステムは、必要に応じてＩ－ｃｏｄｅをｎａｔｉｖｅ　ｃｏｄｅ（バイナリコード、マシンコード）に動的にコンパイルする。また、予めｎａｔｉｖｅｃｏｄｅにコンパイルされているオブジェクトを実行するが、その場合、ＭＫ３１ｂの機能によってＰｅｒｓｏｎａｌｉｔｙオブジェクト３３（バイナリコード生成手段）が呼び出され、アプリケーション（Ａｐｐｌｉｃａｔｉｏｎｓ）３５にサービスを提供する。

【０１２０】図１２に示した核３１を構成するＭＶＭ３１ａおよびＭＫ３１ｂは、その外側がデバイスドライバオブジェクト（Ｄｅｖｉｃｅ　ｄｒｉｖｅｒｓ）３２とＰｅｒｓｏｎａｌｉｔｙオブジェクト（Ｐｅｒｓｏｎａｌｉｔｙ　ｃｏｍｐｏｎｅｎｔ　ｏｂｊｅｃｔｓ）３３によって取り囲まれており、さらに外側には、アプリケーションプログラミングのためのクラスシステム（Ｃｌａｓｓ　Ｌｉｂｒａｒｉｅｓ）３４が用意されており、さらにその外側にはアプリケーション３５が用意されている。

【０１２１】Ｐｅｒｓｏｎａｌｉｔｙオブジェクト３３の層により、Ａｐｅｒｔｏｓシステムは様々なＯＳ（オペレーティングシステム）やＶｉｒｔｕａｌ　Ｍａｃｈｉｎｅ（仮想計算機）を提供することができる。例えば、ｊａｖａプログラムの実行は、ｊａｖａプログラム用のＰｅｒｓｏｎａｌｉｔｙオブジェクトによって、ｊａｖａプログラムがコンパイルされて得られた中間コードであるｊａｖａ　ｂｙｔｅｃｏｄｅが実行されることにより行われる。

【０１２２】Ａｐｅｒｔｏｓシステムは、高度のｐｏｒｔａｂｉｌｉｔｙを実現するために、プログラムをＩ－ｃｏｄｅ（中間コード）にコンパイルし、オブジェクトのメソッドを管理している。しかしながら、Ｉ－ｃｏｄｅは、解釈実行（プログラムを解釈しながら実行すること）を前提に設計されているのではなく、動的コンパイラによってｎａｔｉｖｅ　ｃｏｄｅにコンパイルするように設計されている。しかし、様々な制約により、動的コンパイルが困難である場合、ＭＶＭ３１ａがＩ－ｃｏｄｅを解釈実行する。

【０１２３】しかしながら、ほとんどの場合、Ｉ－ｃｏｄｅは、ｎａｔｉｖｅ　ｃｏｄｅにコンパイルされ、システムを構成するＣＰＵにより直接実行される。従って、Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅによる実行に伴って、実時間性が欠如したり、処理スピードが犠牲になるということはほとんどない。

【０１２４】上記Ｉ－ｃｏｄｅは、高度のＩｎｔｅｒ－Ｏｐｅｒａｂｉｌｉｔｙを実現するため、図２２を参照して後述するように、十分に抽象度の高い２つの命令セット（ＯＰ＿Ｍ，ＯＰ＿Ｒ）から構成されており、そのセマンティクス（意味構造）は、ＭＫ３１ｂのインタフェースと強く関連づけられている。即ち、図１２に示したＡｐｅｒｔｏｓシステムの構造に強く影響を受けた命令セットとされている。これにより、Ａｐｅｒｔｏｓシステムにおいては、ｎａｔｉｖｅ　ｃｏｄｅを前提としていながら、高度のＰｏｒｔａｂｉｌｉｔｙとＩｎｔｅｒ－Ｏｐｅｒａｂｉｌｉｔｙを実現することができる。

【０１２５】次に、ＭＶＭ３１ａとＭＫ３１ｂの仕様について説明する。最初に、ＭＶＭ３１ａが仮定しているデータ構造体と、Ｉ－ｃｏｄｅフォーマットを規定する。

【０１２６】図１３は、図１２に示したＭＶＭ３１ａとＭＫ３１ｂの論理構造を示している。基本的に両者の論理構造は同一であり、ａｃｔｉｖｅ　ｃｏｎｔｅｘｔ４１、ｍｅｓｓａｇｅ　ｆｒａｍｅ４２、およびｅｘｅｃｕｔｉｏｎ　ｅｎｇｉｎｅ４３により構成されているが、ＭＶＭ３１ａは、Ｉ－ｃｏｄｅによる実行をサポートし、ＭＫ３１ｂはｎａｔｉｖｅ　ｃｏｄｅによる実行をサポートしている。

【０１２７】図１３において、ａｃｔｉｖｅ　ｃｏｎｔｅｘｔ４１は、現在実行中のＣｏｎｔｅｘｔ（図１４を参照して後述する）を指しており、ｍｅｓｓａｇｅ　ｆｒａｍｅ４２は、核３１を構成するＭＶＭ３１ａおよびＭＫ３１ｂに対するメッセージ本体を指している。

【０１２８】ＭＫ３１ｂの場合、メッセージ本体の存在する場所は実装方法に依存し、スタックフレームとしてメモリに割り当てられる場合もあるし、ヒープ（ｈｅａｐ）に割り当てられる場合もある。また、いくつかのＣＰＵのレジスタが割り当てられる場合もある。一方、ＭＶＭ３１ａの場合、メッセージフレームは命令コードに続くオペランドを指すことになる。実装によっては、これら以外に内部レジスタ等を必要とするかもしれない

が、それらのレジスタは本仕様とは独立のものである。

【０１２９】図１３において、ｅｘｅｃｕｔｉｏｎ　ｅｎｇｉｎｅ４３は、Ｉ－ｃｏｄｅおよびｎａｔｉｖｅ　ｃｏｄｅを実行する。また、ＭＶＭ３１ａのｅｘｅｃｕｔｉｏｎ　ｅｎｇｉｎｅ４３には、ｐｒｉｍｉｔｉｖｅ　ｏｂｊｅｃｔｓ４４が含まれており、これは、図１８を参照して後述するプリミティブオブジェクトを処理するコードである。

【０１３０】図１４は、ＣｏｎｔｅｘｔとＤｅｓｃｒｉｐｔｏｒの論理構造を示している。プログラムの１つの実行状態を示すＣｏｎｔｅｘｔ構造体５１は、ｏｂｊｅｃｔ、ｃｌａｓｓ、ｍｅｔｈｏｄ、ｍｅｔａなどのフィールドを有し、このうち、ｍｅｔｈｏｄには、ｉｃｏｄｅとｍｉｎｆｏのフィールドがさらに設けられている。このＣｏｎｔｅｘｔ構造体５１は、ＭＶＭ３１ａの状態を保持し、ＣＰＵのレジスタに相当し、メモリ管理やプログラム間の通信方法などとは完全に独立している。

【０１３１】Ｃｏｎｔｅｘｔ構造体５１は、Ｃｏｎｔｅｘｔプリミティブオブジェクトであり、図１６を参照して後述するように、その各フィールドは所定の情報にリンクしている。また、Ｃｏｎｔｅｘｔ構造体５１は、核３１（ＭＶＭ３１ａおよびＭＫ３１ｂ）の実装に深く依存しているが、図１４においては、主に依存しない部分を示している。

【０１３２】このＣｏｎｔｅｘｔ構造体５１において重要なフィールドは、ｍｅｔａフィールドであり、Ｄｅｓｃｒｉｐｔｏｒ構造体５２を指している。Ｄｅｓｃｒｉｐｔｏｒ構造体５２のエントリは、＃ｔａｇ、ｃｏｎｔｅｘｔ、およびｓｅｌｅｃｔｏｒの3つの組みであり、このＤｅｓｃｒｉｐｔｏｒ構造体５２によって、Ｐｅｒｓｏｎａｌｉｔｙオブジェクト３３のＡＰＩ（アプリケーションプログラミングインタフェース）が決定される。＃ｔａｇはＡＰＩの名称を、ｃｏｎｔｅｘｔとｓｅｌｅｃｔｏｒでＡＰＩのアドレスをそれぞれ表している。

【０１３３】図１５は、ＭＶＭ３１ａの全体構造を示している。基本的に、全ての必要な情報は、Ｃｏｎｔｅｘｔ構造体５１からリンクを辿ることにより参照することができるようになされている。即ち、Ｃｏｎｔｅｘｔ構造体５１が、ｏｂｊｅｃｔ６１にリンクされ、このｏｂｊｅｃｔ６１は、ｏｂｊｅｃｔ６１に対応するクラスへのリンク（ｃｌａｓｓ　ｐｏｉｎｔｅｒ）とインスタンス領域（ｏｂｊｅｃｔｄｅｐｅｎｄｅｎｔ　ｆｉｅｌｄ）より構成される。インスタンス領域にどのような情報が格納されるか、あるいはそのレイアウトは、オブジェクトの実装に依存する。

【０１３４】ｃｌａｓｓ６２は、主としてメソッドを保持する。ｃｌａｓｓ６２はその名称（ｃｌａｓｓ　ｎａｍｅ）、その実装に依存する部分（ｃｌａｓｓ　ｄｅｐｅｎｄｅｎｔ　ｆｉｅｌｄ）、およびＩ－ｍｅｔｈｏｄ構造体６３へのリンクテーブル（ｍｅｔｈｏｄ　ｔａｂｌｅ）により構成される。Ｉ－ｍｅｔｈｏｄ構造体６３は、Ｂｌｏｃｋプリミティブオブジェクトであり、ヘッダ（Ｈｅａｄｅｒ）、Ｉ－ｃｏｄｅ部、および変数テーブル（Ｖａｒｉａｂｌｅ　ｔａｂｌｅ）からなる。ｍａｇｉｃ＃は、ＭＶＭ３１ａの管理番号（ＩＤ）である。基本的に、Ｉ－ｃｏｄｅ命令のオペランドは、この変数テーブルエントリを介して目的オブジェクトを参照する。

【０１３５】図１５において、グレーの部分（Ｃｏｎｔｅｘｔ５１、Ｉ－ｍｅｔｈｏｄ６３）はＭＶＭ３１ａに依存する部分であり、ＭＶＭ３１ａがＩ－ｃｏｄｅを解釈実行する場合に必要とする構造体である。

【０１３６】図１６は、Ｃｏｎｔｅｘｔ構造体５１を中心に、そこからリンクされている構造体を図示したものである。Ｃｏｎｔｅｘｔ構造体５１のオブジェクトフィールドは、ｏｂｊｅｃｔ６１を指し、ｃｌａｓｓフィールドは、ｃｌａｓｓ６２のｃｌａｓｓ　ｄｅｐｅｎｄｅｎｔ　ｆｉｅｌｄを指している。また、ｍｅｔｈｏｄフィールドはＩ－ｍｅｔｈｏｄ６３を指しており、ｍｅｔｈｏｄフィールドのｉｃｏｄｅは、Ｉ－ｍｅｔｈｏｄ６３のＩ－ｃｏｄｅを指している。ｉｃｏｄｅは、プログラムカウンタに相当し、実行中のプログラムを指す。ｍｅｔｈｏｄフィールドのｖｔａｂｌｅは、Ｉ－ｍｅｔｈｏｄ６３のｖａｒｉａｂｌｅ　ｔａｂｌｅを指している。

【０１３７】ｔｅｍｐｏｒａｒｙフィールドは、一時データ保存領域を指し、ｍｅｔａフィールドは、上述したように、ｄｅｓｃｒｉｐｔｏｒ５２を指している。また、オブジェクト６１のｃｌａｓｓ　ｐｏｉｎｔｅｒは、ｃｌａｓｓ６２を指し、ｃｌａｓｓ６２のｍｅｔｈｏｄ　ｔａｂｌｅは、Ｉ－ｍｅｔｈｏｄ６３を指している。

【０１３８】ｖａｒｉａｂｌｅ　ｔａｂｌｅエントリは、ｔｙｐｅとｖａｌｕｅの組みであり、ｖａｌｕｅはｔｙｐｅに依存する。

【０１３９】ＭＶＭ３１ａは、図１７に示したタイプを処理するようになされており、ｔｙｐｅがＴ＿ＰＲＩＭＩＴＩＶＥのとき、ｖａｌｕｅ欄はプリミティブオブジェクトを参照する。この場合、ｖａｌｕｅ欄には、例えば、＜Ｐ＿ＩＮＴＥＧＥＲ，ｉｍｍｅｄｉａｔｅ＞，＜Ｐ＿ＳＴＲＩＮＧ，ａｄｄｒｅｓｓ　ｔｏ　ｈｅａｐ＞といった、図１８に示すような＜Ｐ＿ＣＬＡＳＳ，Ｐ＿ＢＯＤＹ＞の組が格納される。図１８は、プリミティブオブジェクトの一覧と、そのインタフェース名を示している。図１９乃至図２１は、図１８に示したプリミティブオブジェクトのインタフェースを、Ｉｎｔｅｒｆａｃｅ　Ｄｅｆｉｎｉｔｉｏｎ　Ｌａｎｇｕａｇｅ（ＩＤＬ）法により記述した例を示している。このＩＤＬ法については、「ＣＯＲＢＡ　Ｖ２．０，Ｊｕｌｙ　１９９

５，Ｐ３－１乃至３－３６」に開示されている。

【０１４０】また、図１７において、ｔｙｐｅがＴ＿ＰＯＩＮＴＥＲのとき、オブジェクトを参照するためのＩＤがｖａｌｕｅ欄に格納される。このＩＤは、その位置が独立な、システムで唯一の値であり、Ｐｅｒｓｏｎａｌｉｔｙオブジェクト３３によって、そのＩＤに対応するオブジェクトの位置が特定される。

【０１４１】図２２は、ＭＶＭ３１ａが解釈実行する２つの命令セットを示している。命令セットＯＰ＿Ｍは、図１２において、外側から内側に入る命令であり、命令セットＯＰ＿Ｒは、内側から外側に戻る命令である。

【０１４２】即ち、命令セットＯＰ＿Ｍにより、その第１オペランドで示されたオペレーションが実行される。このオペレーションは、Ｐｅｒｓｏｎａｌｉｔｙオブジェクト３３によって処理されるか、またはプリミティブオブジェクトによって処理される。コンパイラは、より効率的にアプリケーション３５を実行するために、いくつかの処理をプリミティブオブジェクトによって処理するように、Ｉ－ｃｏｄｅを生成する。これにより、例えば、Ｉｎｔｅｇｅｒの算術演算は、図１８の先頭のプリミティブオブジェクトによって処理される。

【０１４３】次に、ＭＫ３１ｂのインタフェースを規定する。図２３は、マイクロカーネル（ＭＫ）３１ｂのインタフェースを示している。ＭＶＭ３１ａとＭＫ３１ｂの論理構造は、図１３に示したように同様であり、ＭＫ３１ｂは、図２２に示した命令セットＯＰ＿ＭおよびＯＰ＿Ｒをシステムコールとして処理する。

【０１４４】上述したような構成のＡｐｅｒｔｏｓシステムを、例えば図４に示したクライアント２（２－１，２－２）に適用することができる。そして、図１２に示した核３１以外のもののうち、任意のものをサーバ１からダウンロードすることにより、所定のアプリケーションを実行するための最適な実行環境をクライアント２上に構築することができる。

【０１４５】上述したように、ここでダウンロードするオブジェクトとしては次のものが考えられる。

【０１４６】（１）すべてのアプリケーションプログラム

【０１４７】（２）クライアントが備えるハードウェア資源を制御するためのデバイスドライバ群（例えば、ＭＰＥＧドライバ、ＡＴＭドライバ、画像制御ドライバ等）

【０１４８】（３）アプリケーションプログラムに対してシステムサービスを提供するオブジェクト群（ｐｅｒｓｏｎａｉｔｙオブジェクト）（例えば、ＶＣＲコマンド管理、ストリーム管理、実時間スケジューラ、メモリ管理、ウィンドウ管理、ダウンロード制御、通信プロトコル管理、実行管理等）

【０１４９】これらを組み合わせて、アプリケーションプログラムに対して最適な実行環境をクライアント上に構築することができる。

【０１５０】例えば、ｊａｖａプログラムを実行させたい場合、ｊａｖａ用のｐｅｒｓｏｎａｌｉｔｙオブジェクト３３と、ｊａｖａプログラム（アプリケーション）３５をサーバ１よりダウンロードする。このｐｅｒｓｏｎａｌｉｔｙオブジェクト３３により、Ａｐｅｒｔｏｓシステムは、Ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅを提供することができる。ダウンロードされたｊａｖａプログラムは、コンパイラにより一旦ｊａｖａ　ｂｙｔｅｃｏｄｅと呼ばれる中間コードにコンパイルされ、上記ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅにより実行される。あるいは、上述したように、ｎａｔｉｖｅ　ｃｏｄｅにコンパイルした後、実行するようにすることも可能である。

【０１５１】以上の実施例においては、サーバからクライアントに所定のオブジェクトをダウンロードするようにしたが、所定のクライアントから、所定のサーバにオブジェクトをダウンロードしたり、あるいはサーバ同士、クライアント同士でオブジェクトをダウンロードする場合にも、本発明を適用することが可能である。

【０１５２】また、上記実施例においては、Ｉ－ｃｏｄｅを２つの命令により構成するようにしたが、これに限定されるものではない。

【０１５３】

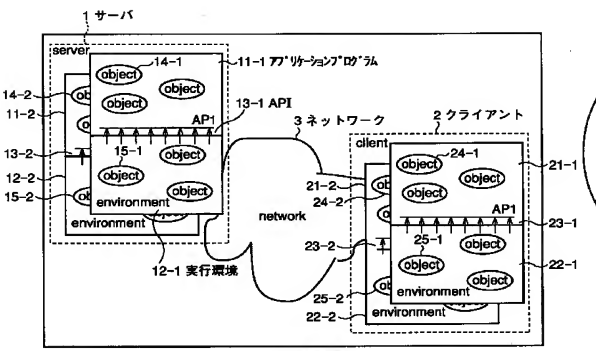
【発明の効果】以上の如く請求項１記載のデータ処理方法および請求項９に記載のデータ処理装置によれば、クライアントがダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査し、その検査結果に対応して、アプリケーションプログラムをクライアントにダウンロードするようにしたので、クライアントの構成を簡略化し、低コスト化するとともに、低コストでアプリケーションプログラムを提供することが可能となる。

【０１５４】請求項１０に記載のデータ処理装置によれば、サーバに対してダウンロードするアプリケーションプログラムを実行環境に関する告知を行い、その告知に対応して、サーバからアプリケーションプログラムをダウンロードするようにしたので、構成が簡単で低コストの装置を実現することが可能となる。また、低コストでアプリケーションプログラムを提供することができる。

【０１５５】請求項１１に記載のデータ処理装置、および請求項１５に記載のデータ処理方法によれば、中間コードに変換されたアプリケーションプログラムを解釈し、実行するか、または、中間コードを動的にコンパイルし、生成されたバイナリコードを実行することで、アプリケーションプログラムを実行するようにしたので、動的コンパイルが困難な場合、中間コードを逐次解釈し、実行することができる。また、中間コードを簡単な構成とすることにより、ポータブルなアプリケーションを構築することができる。

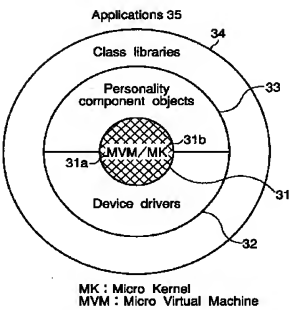
【図面の簡単な説明】
【図１】本発明のデータ処理方法を適用するシステムの構成例を示す図である。
【図２】アプリケーションプログラムの構成を示す図である。
【図３】並行オブジェクトの構成を示す図である。
【図４】サーバから複数ベンダのクライアントに対するオブジェクトのダウンロードを説明する図である。
【図５】インクリメンタルダウンロードを説明する図である。
【図６】メタ標準を説明する図である。
【図７】オブジェクトの動的変更を説明する図である。
【図８】オブジェクトの動的拡張を説明する図である。
【図９】クライアントの最小機能を説明する図である。
【図１０】アプリケーションに適したクライアント環境の構築とその動的再構成を説明する図である。
【図１１】フィーチャストラクチャの構成を説明する図である。
【図１２】本発明のデータ処理装置を応用したＡｐｅｒｔｏｓシステムの構成例を示す図である。
【図１３】ＭＶＭとＭＫの論理構造を示す図である。
【図１４】ＣｏｎｔｅｘｔとＤｅｓｃｒｉｐｔｏｒの論理構造を示す図である。
【図１５】ＭＶＭの全体構造を示す図である。
【図１６】Ｃｏｎｔｅｘｔとその回りのデータ構造体を示す図である。
【図１７】Ｖａｒｉａｂｌｅ　ｔａｂｌｅエントリのｔｙｐｅを示す図である。
【図１８】プリミティブオブジェクトの一覧とそのインタフェース名を示す図である。
【図１９】プリミティブオブジェクトのインタフェースを示す図である。
【図２０】プリミティブオブジェクトのインタフェースを示す図である。
【図２１】プリミティブオブジェクトのインタフェースを示す図である。
【図２２】Ｉ－ｃｏｄｅ命令セットを示す図である。
【図２３】ＭＫのインタフェースを示す図である。
【符号の説明】
１　サーバ，２　クライアント，３　ネットワーク，１１　アプリケーションプログラム，１２　実行環境，１３　アプリケーションプログラムインタフェース，１４，１５　オブジェクト，２１　アプリケーションプログラム，２２　実行環境，２３　アプリケーションプログラムインタフェース，３１　核，３１ａＭＶＭ，３１ｂ　ＭＫ，３２　Ｄｅｖｉｃｅ　ｄｒｉｖｅｒｓ，３３　Ｐｅｒｓｏｎａｌｉｔｙ　ｃｏｍｐｏｎｅｎｔ　ｏｂｊｅｃｔ，３４　Ｃｌａｓｓ　ｌｉｂｒａｒｉｅｓ，３５　Ａｐｐｌｉｃａｔｉｏｎｓ，４１　ａｃｔｉｖｅ　ｃｏｎｔｅｘｔ，４２　ｍｅｓｓａｇｅ　ｆｒａｍｅ，４３　ｅｘｅｃｕｔｉｏｎｅｎｇｉｎｅ，４４　ｐｒｉｍｉｔｉｖｅ　ｏｂｊｅｃｔｓ，５１　ｃｏｎｔｅｘｔ，５２　Ｄｅｓｃｒｉｐｔｏｒ，６１　ｏｂｊｅｃｔ，６２　ｃｌａｓｓ，６３　Ｉ－ｍｅｔｈｏｄ

【図１】



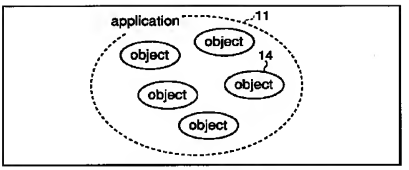
システム構成例

【図１２】



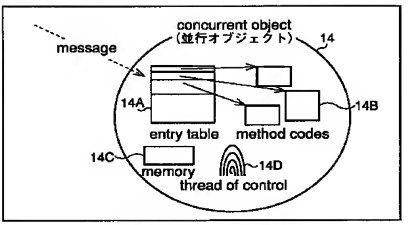
Apertosシステムの機能構成図

【図２】



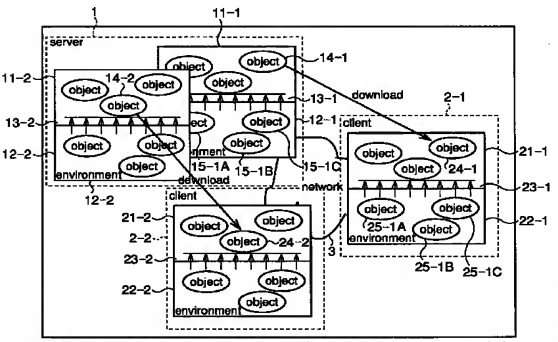
アプリケーションの構成

【図３】



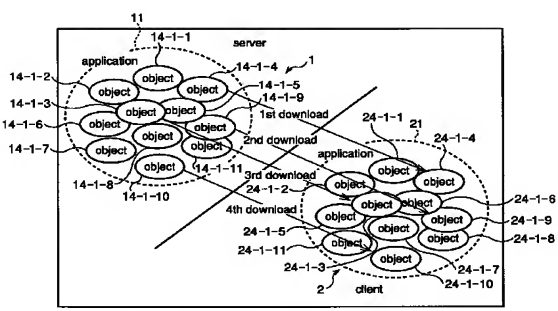
並行オブジェクトの構造

【図４】



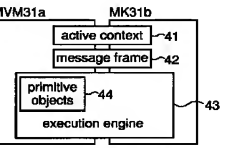
複数ベンダのクライアントにサーバからオブジェクトをダウンロードする方法

【図５】



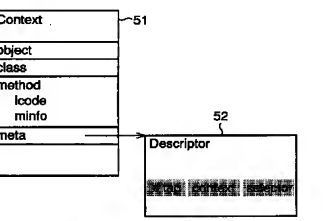
インクリメンタルダウンローディング

【図１３】



MVMとMKの論理的構造

【図１４】



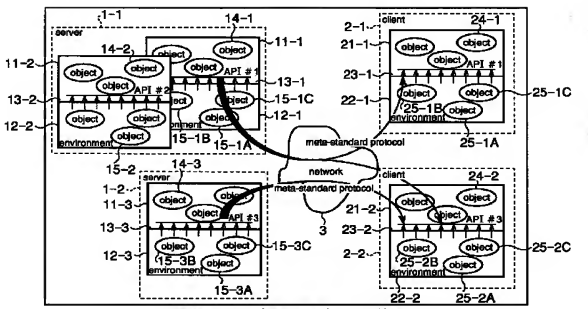
ContextとDescriptorの論理構造

【図１７】

type	description
T_INVALID	invalid entry
T_PRIMITIVE	The value field denotes a primitive object, which is a dependent object on MVM
T_POINTER	The value field contains a pointer to an object

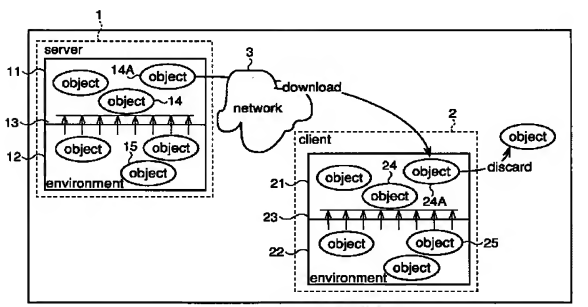
variable tableエントリのtype

【図6】



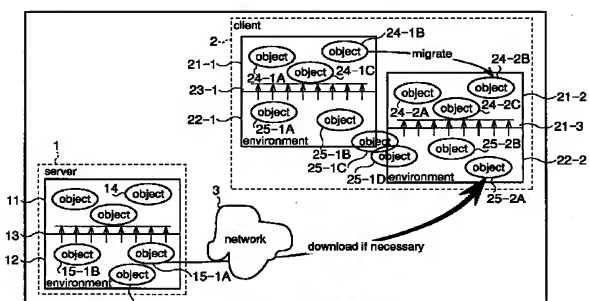
メタ標準による最小限の標準化とその拡張

【図7】



オブジェクトの動的変更

【図8】



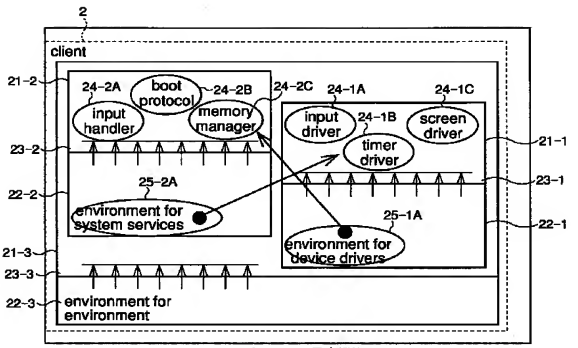
オブジェクトの動的拡張

【図19】

```
19-1
struct Message {
        long length;
        any body [ ];
};

exception outrange (long position);
exception overflow ( );
exception undef (int primitive; int selector);
exception zerodiv ( );

interface Integer {
        Integer operator + (in Integer value)
                raise (overflow);
        Integer operator − (in Integer value)
                raise (overflow);
        Integer operator * (in Integer value);
        Integer operator ／ (in Integer value)
                raise (zerodiv);
        Integer remainder (in Integer value);
        Integer and (in Integer value);
        Integer or (in Integer value);
        Integer xor (in Integer value);
        Integer not (void);
        Integer operator = (in Float value);
        Integer operator = (in DoubleFloat value);
        Boolean operator < (in Integer value);
        Boolean operator <= (in Integer value);
        Boolean operator == (in Integer value);

};
interface Float {
        Float operator + (in Float value)
                raise (overflow, underflow);
        Float operator − (in Float value)
                raise (overflow, underflow);
        Float operator * (in Float value)
                raise (overflow, underflow);
        Float operator ／ (in Float value)
                raise (overflow, underflow,zerodiv);
        Float operator = (in Integer value);
        Float operator = (in DobleFloat value);
                raise (overflow);
        Boolean operator < (in Float value);
        Boolean operator <= (in Float value);
        Boolean operator == (in Float value);
```

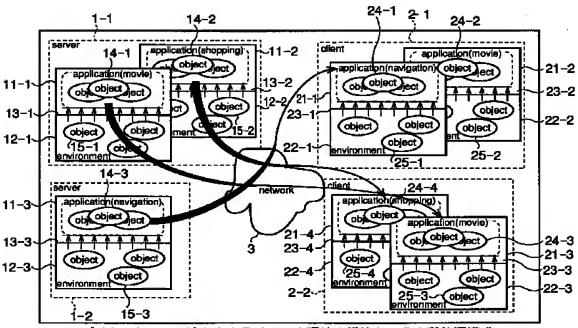
プリミティブオブジェクトインターフェース

【図９】



クライアントの最小機能

【図１０】



アプリケーションに適したクライアント環境の構築と、その動的再構成

【図２２】

name	description
OP_M	execute the operation specified by the first argument. This operation is processed either by personality or by a primitive object.(operands:operation,number of operands,...)
OP_R	give control back to the object when the operation requested by OP_M returns (operands:target,return arguments)

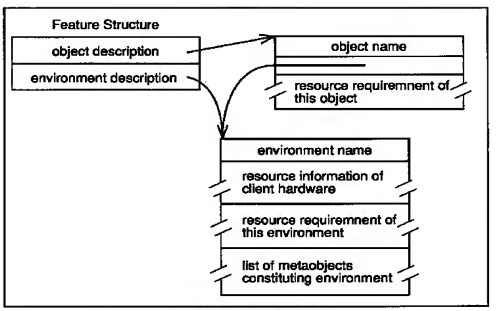
I-code 命令セット

【図２３】

```
Interface MetaCore {
    mcError M (in long method, in Message msg);
    mcError R (in CName cbt, in long method, in Message msg);
```

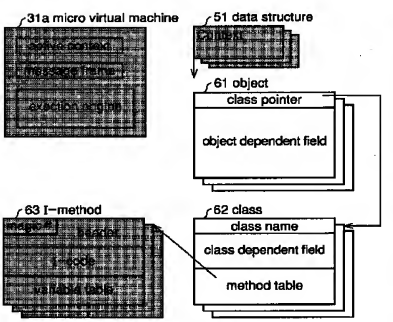
Micro Kernelのインターフェース

【図１１】



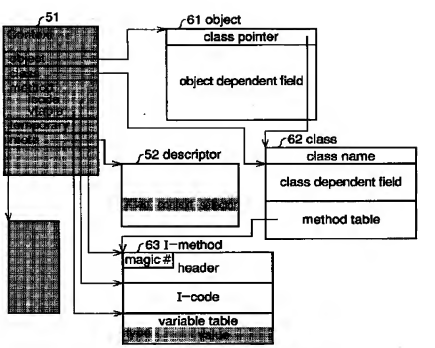
Feature Structureの構造

【図１５】



MVM全体構造

【図１６】



Contextとその回りのデータ構造体

【図18】

class	P_CLASS	P_BODY	description
Integer	P_INTEGER	immediate	integer number
Float	P_FLOAT	immediate	IEEE 754 32 bits floating point number
Double Float	P_DFLOAT	immediate	IEEE 754 64 bits floating point number
Boolean	P_BOOLEAN	immediate	TRUE or FALSE
String	P_STRING	address to heap	sequence of UNICODE characters
Array	P_ARRAY	address to heap	sequence of data of the same type
Context	P_CONTEXT	CName	representation of object execution
Jump	P_JUMP	I−code index	target address to which execution resumes
Pointer	P_POINTER	address to memory	address
Assign	P_ASSIGN	n/a	duplication of a variable entry
Mailer	P_MAILER	n/a	invocation of a method
Undef	P_UNDEF	n/a	undefined primitive object

プリミティブオブジェクト

【図21】

```
19−3
interface Pointer {
        void put (in Address pos, in byte value)
                exception outrange;
        byte get (in Address pos)
                exception outrange;
        void put (in Address pos, in word value)
                exception outrange;
        word get (in Address pos)
                exception outrange;
        void put (in Address pos, in long value)
                exception outrange;
        long get (in Address pos)
                exception outrange;
        void put (in Address pos, in longlong value)
                exception outrange;
        longlong get (in Address pos)
                exception outrange;
};
interface Assign {
        void operator = (in int position1, in int position2);
};
interface Mailer {
        void call (in ID target, in Integer select, in Message msg);
        void reply ( );
};
interface undef
};
```

プリミティブオブジェクトインターフェース(続き)

【図20】

```
19−2
interface DoubleFloat {
        DoubleFloat operator + (in DoubleFloat value)
                raise (overflow, underflow);
        DoubleFloat operator − (in DoubleFloat value)
                raise (overflow, underflow);
        DoubleFloat operator * (in DoubleFloat value)
                raise (overflow, underflow);
        DoubleFloat operator ／ (in DoubleFloat value)
                raise (overflow, underflow,zerodiv);
        DoubleFloat operator = (in Integer value)
                raise ( );
        DoubleFloat operator = (in Float value)
                raise ( );
        Boolean operator < (in DoubleFloat value)
                raise ( );
        Boolean operator <= (in DoubleFloat value)
                raise ( );
        Boolean operator == (in DoubleFloat value)
                raise ( );
};
interface Boolean {
        Boolean not (void);
};
interface String {
        String operator+ (in String string);
        String substring (in Integer position, in Integer length);
                raise (outrange);
        Integer length(void);
};
interface Array {
        void put (in Integer Index, in any value);
                raise (outrange);
        any get (in Integer Index)
                raise (outrange);
        Integer length(void);
};
interface Context {
        ...accessers...
};
interface Jump {
        void eval (void)
                raise (outrange);
        void evalT (Boolean cond)
                raise (outrange);
        void evalF (Boolean condition)
                raise (outrange);
};
```

プリミティブオブジェクトインターフェース(続き)